The Patent Office
PATENTS • DESIGNS • COPYRIGHT • TRADE MARKS

INVESTOR IN PEOPLE

ARM Limited
% D Young & Co
21 New Fetter Lane
LONDON
EC4A 1DA

The Patent Office
Patents Directorate

Concept House
Cardiff Road, Newport
South Wales NP10 8QQ

Examiner: 01633 814493
†**E-mail:** Pierre.Oliviere@patent.gov.uk
Switchboard: 01633 814000
Fax: 01633 814444
Minicom: 08459 222250
DX 722540/41 Cleppa Park 3
http://www.patent.gov.uk

Your Reference: P014638GB
Application No: GB 0221772.7

22 April 2003

Dear Sirs

Patents Act 1977: Search Report under Section 17(5)

I enclose two copies of my search report and a copy of the citations.

Publication

I estimate that, provided you have met all formal requirements, preparations for publication of your application will be completed soon after **10 February 2004**. You will then receive a letter informing you of completion and telling you the publication number and date of publication.

Amendment/withdrawal

If you wish to file amended claims for inclusion with the published application, or to withdraw the application to prevent publication, you must do so before the preparations for publication are completed. **No reminder will be issued.** If you write to the Office less than 3 weeks before the above completion date, please mark your letter prominently: **"URGENT - PUBLICATION IMMINENT".**

Yours faithfully

Pierre Oliviere
Examiner

---

†**Use of E-mail:** Please note that e-mail should be used for correspondence only.

Application No: GB 0221772.7
Claims searched: 1-65
Examiner: Pierre Oliviere
Date of search: 22 April 2003

Patents Act 1977 : Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance
X	1, 22 and 43 at least	US 6425070 B1 (ZOU) See columns 2 and 10 to 12.
X	1, 22 and 43 at least	US 5673410 A (KURISU) See figure 8 and columns 1 to 3.
X	1, 22 and 43 at least	EP 1102065 A1 (TEXAS) See figure 7B and related description.

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC[V]:

G4A

Worldwide search of patent documents classified in the following areas of the IPC[7]:

G06F

The following online and other databases have been used in the preparation of this search report:

WPI, EPODOC, PAJ

INVESTOR IN PEOPLE

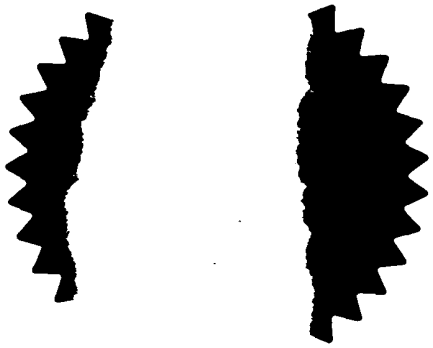
The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.



Signed

Dated 22 May 2003

Patents Form 1/77

Patents Act 1977
(Rule 16)

The Patent Office

20SEP02 E749612-1 002246
P01/7700 0.00-0221772.7

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

THE PATENT OFFICE
J
19 SEP 2002
LONDON

The Patent Office

Cardiff Road
Newport
South Wales
NP10 8QQ

1. Your reference
   P014638GB

2. Patent application number
   (The Patent Office will fill in this part)
   0221772.7

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*
   ARM Limited
   110 Fulbourn Road
   Cherry Hinton
   Cambridge
   CB1 9NJ

   Patents ADP number *(if you know it)*
   7498124002

   If the applicant is a corporate body, give the country/state of its incorporation
   United Kingdom

4. Title of the invention
   Executing Variable Length Instructions Stored Within a Plurality of Discrete Memory Address Regions

5. Name of your agent *(if you have one)*
   D Young & Co

   "Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*
   21 New Fetter Lane
   London
   EC4A 1DA

   Patents ADP number *(if you know it)*
   59006

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*
   a) any applicant named in part 3 is not an inventor, or
   b) there is an inventor who is not named as an applicant, or
   c) any named applicant is a corporate body.
   See note (d))
   Yes

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	0
Description	14
Claim *(s)*	10
Abstract	1
Drawing *(s)*	5

10. If you are also filing any of the following, state how many against each item.

Priority documents	0
Translations of priority documents	0
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	4
Request for preliminary examination and search *(Patents Form 9/77)*	1
Request for substantive examination *(Patents Form 10/77)*	0
Any other documents *(please specify)*	0

11. I/We request the grant of a patent on the basis of this application.

Signature D Young & Co (Agents for the Applicants) Date 19 September 2002

12. Name and daytime telephone number of person to contact in the United Kingdom: Nigel Robinson 023 8071 9500

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.

d) If you have answered 'Yes' Patents Form 7/77 will need to be filed.

e) Once you have filled in the form you must remember to sign and date it.

f) For details of the fee and ways to pay please contact the Patent Office.

Patents Form 7/77

Patents Act 1977
(Rule 15)

Statement of inventorship and of right to grant of a patent

The Patent Office

Cardiff Road
Newport
South Wales
NP10 8QQ

1.	Your reference	P014638GB
2.	Patent application number *(if you know it)*	0221772.7
3.	Full name of the or of each applicant	ARM Limited
4.	Title of the invention	Executing Variable Length Instructions Stored Within a Plurality of Discrete Memory Address Regions
5.	State how the applicant *(s)* derived the right from the inventor *(s)* to be granted a patent	By Virtue of Employment
6.	How many, if any, additional Patents Forms 7/77 are attached to this form? *(see note (c))*	0
7.		I/We believe that the person *(s)* named over the page *(and on any extra copies of this form)* is/are the inventor *(s)* of the invention which the above patent application relates to. Signature DY&Co Date 19 Sept 2002 D Young & Co (Agents for the Applicants)
8.	Name and daytime telephone number of person to contact in the United Kingdom	Nigel Robinson 023 8071 9500

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there are more than three inventors, please write the names and addresses of the other inventors on the back of another Patents Form 7/77 and attach it to this form.

d) When an application does not declare any priority, or declares priority from an earlier UK application, you must provide enough copies of this form so that the Patent Office can send one to each inventor who is not an applicant.

e) Once you have filled in the form you must remember to sign and date it.

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Surname: FRANCIS

First Names: Hedley James

Hill Croft
Brinkley Road
Carlton
Newmarket
Suffolk, CB8 9JY

7882954002

Patents ADP number *(if you know it):*

Surname: PIRY

First Names: Frederic Claude Marie

11, chemin des Plateaux Fleuris
06800 Cagnes-sur-Mer
France

8468324001

Patents ADP number *(if you know it):*

Surname: BROYER

First Names: Pierre Michel

"Le Pont Royal"
84, avenue Matisse
06140 Vence
France

8468332001

Patents ADP number *(if you know it):*

Reminder

Have you signed the form?

EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

This invention relates to the field of data processing systems. More particularly, this invention relates to data processing systems operable to execute variable length instructions stored within a plurality of memory address regions, e.g. a program fragmented within the memory.

It is known to provide data processing systems which execute variable length instructions. An example of such systems are the Jazelle enabled processors produced by ARM Limited. These processors are capable of executing programs formed of Java bytecodes that vary in their byte length. It is desirable within such systems in some circumstances to allow a program formed of variable length instructions to be stored in a plurality of different memory regions, i.e. for the program to be fragmented in memory. In a secure system such fragmentation may be desirable as a way of obfuscating the computer program concerned. In other circumstances it may be desirable in order to better use of the memory available by filling all portions of that memory even if they are discrete.

Figure 1 of the accompanying drawings illustrates a problem which can arise in such systems. In particular, Figure 1 illustrates a variable length program instruction which is three bytes in length and comprises a bytecode BCX followed by two operands OpdXOpdX'. This 3-byte instruction spans two discrete memory regions with the first byte being in a current memory region and the second and third bytes being in a following memory region. Standard hardware for executing such variable length instructions assumes all the bytes of a variable length instruction will be found at sequentially adjacent memory addresses. In the case illustrated in Figure 1, the standard hardware would assume that the second and third bytes of the variable length instruction immediately followed the first byte of the variable length instruction, whereas in fact they are in a discrete memory region separated from the first byte.

The present invention recognises the problems associated with supporting variable length instructions of a program which is stored in a fragmented manner within the memory as well as providing a solution to these problems.

Viewed from one aspect the present invention provides a method of executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of a data processing apparatus, said method comprising the steps of:

detecting an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region;

concatenating instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

diverting program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

restoring program execution flow to execute instructions following said variable length instruction from within said following memory address region.

The invention provides the solution of concatenating the instruction data from portions of the discrete memory regions containing the variable length instruction which spans the boundary and storing this concatenated data within a fix-up memory region. The variable length instruction concerned can then be executed from its location within this fix-up region following which program flow can be returned to the normal memory region following the gap between the discrete memory regions. This provides a mechanism for dealing with fragmented program code comprising variable length instructions for which the start address of the instruction following that spanning the gap will not be known until the instruction spanning the gap has been executed whilst keeping the software overhead needed to support such a capability at a reduced level. More particularly, in preferred embodiments of the invention the step

of detecting can be performed under hardware control with the steps of concatenating, diverting and restoring performed under software control, whilst the actual execution of the variable length instruction spanning the gap can be performed in hardware.

The present invention is particularly well suited for use in systems in which the variable length instructions are fetched from memory to an instruction buffer before being executed. Such an arrangement is convenient as it permits the identification of an instruction spanning discrete memory regions to be made as it is fetched from memory to the instruction buffer and the detection of the attempt to execute that instruction to be made as an attempt is made to read the variable length instruction from the instruction buffer.

More particularly, in preferred embodiments the hardware for the instruction buffer is simplified by providing that it normally operates to fetch instruction data from sequential memory addresses and identifies buffer memory locations extending beyond the endpoint of a current memory address region as containing invalid data such that if an attempt is made to execute from such buffer memory locations then appropriate corrective action may be taken.

The mechanisms for diverting and restoring program flow may be conveniently provided in preferred embodiments by manipulating a program counter value used to specify a next program instruction to be executed.

Control of the execution of variable length instructions from the fix-up memory region is conveniently provided in preferred embodiments using a single step flag which is set before program flow is diverted to the fix-up memory region and serves to allow only a single instruction to be executed from that fix-up memory region before control is returned to software for the step of restoring program flow whilst at the same time the single step flag is preferably cleared by the hardware.

Whilst it will be appreciated that the single step flag may be stored in a variety of different places, it is conveniently provided within a coprocessor register which may

be accessed under program control by the software which is responsible for diverting program flow to the fix-up memory region.

A problem with variable length instructions spanning a gap between discrete memory regions is that the start point of the next instruction to be executed is not known until the instruction spanning the gap has been at least decoded and accordingly its length determined. Preferred embodiments of the invention deal with this by calculating the memory address of the following variable length instruction within the following memory region using as inputs the start address of the following memory region and the program counter value pointing to the following instruction within the fix-up memory region as determined after the current variable length instruction has been executed from within the fix-up memory region. In order to conveniently support this operation preferred embodiments serve to store the start address of the following memory region before program flow is diverted to the fix-up memory region such that this information will be available for use in calculating the entry point for use in the restoring step as program flow is restored to the correct point within the following memory region.

Whilst it will be appreciated that the present technique may be used with a variety of variable length instruction types, the technique is particularly useful when the variable length instructions are Javacard bytecode instructions to be executed as native instructions as the more abstract nature of these instructions is such that it is more likely that they may be fragmented within the memory of the physical system. In order to deal with fragmentation of Javacard bytecode programs it is desirable that the steps of concatenating, diverting and restoring are performed under software control using a further instruction set which may be executed by the data processing apparatus. Thus, whilst the data processing apparatus may support Javacard bytecodes, the physical level management of issues such as memory addressing and memory access writes and the like may be controlled at a lower operating system using software written in a different instruction set more closely related to the physical hardware concerned.

In the context of a system supporting execution of Javacard bytecodes with a further instruction set, the steps of diverting and restoring may conveniently be performed by state switching branch instructions that serve to switch execution to a Javacard mode starting from a specified memory address location which can be conveniently manipulated to control program flow.

Viewed from another aspect the present invention provides apparatus for executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory, said apparatus comprising:

a detector operable to detect an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region;

combining logic operable to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

diverting logic operable to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

restoring logic operable to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

Viewed from a further aspect the invention provides a computer program product for controlling a data processing apparatus operable to executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of said data processing apparatus, said computer program product comprising:

code operable after an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being

a current memory address region and a following memory address region, said code including:

(i) concatenating code operable to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

(ii) diverting code operable to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

(iii) restoring code operable to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

An embodiment of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 schematically illustrates a variable length instruction spanning discrete memory regions;

Figure 2 is a flow diagram illustrating control of an instruction buffer used to buffer variable length instructions being fetched from a memory as instruction data before those variable length instructions are decoded;

Figures 3A and 3B are a flow diagram illustrating how processing may be performed when a variable length instruction spanning discrete memory regions is encountered;

Figure 4 is a diagram illustrating the operations described in relation to Figures 3A and 3B in a different way; and

Figure 5 is a schematic hardware diagram illustrating a system in which the techniques described in relation to Figures 2 to 4 may be used.

Figure 2 illustrates a flow diagram describing how an instruction decoder buffer may be controlled within a data processing system such as that described in PCT Published Patent Application WO-A-02/29507 (reference may be made to this published document for a description of a native Java processor). In particular, the instruction decoder buffer waits at step 2 until a slot becomes available to be refilled with instruction data fetched from the memory. When such a slot becomes available processing proceeds to step 4 at which the next 32-bit instruction data word is fetched. Since the Javacard bytecode instructions (Javacard is a special version of Java tailored for use with smartcards) contained within the instruction data word are a variable length, it will be appreciated that such a 32-bit instruction data word may contain one or more Javacard bytecode instructions or parts of Javacard bytecode instructions.

At step 6, a determination is made as to whether the fetch initiated at step 4 produced a memory abort. Such an abort may occur for a variety of reasons, but typical reasons may be a protection error or the crossing of a memory page table boundary or the like (e.g. due to fragmentation).

If a memory abort does not occur, then processing proceeds to step 8 at which the instruction data word fetched is stored within the available slot in the instruction decoder buffer and processing returned to step 2.

If a memory abort was detected at step 6, then processing proceeds to step 10 at which the slot in the instruction decoder buffer is marked with aborted memory fetch flags and then processing returned to step 2. Such aborted memory fetch flags serve the function of indicating to the Javacard bytecode decoder should it seek to access that instruction data that the instruction data is not valid since an abort occurred when the fetch attempt was made and accordingly a memory abort handler routine should be initiated. The memory abort handler is only triggered if an attempt is made to execute bytecodes which were subject to a memory abort.

Figures 3A and 3B are a flow diagram illustrating processing which occurs in order to deal with variable length instructions that span a gap in the memory address space. At step 12 the Javacard decoder reads the bytes for the next instruction from the instruction decode buffer. Step 14 identifies if any of these bytes are marked with a memory abort flag. If none of the bytes are marked with a memory abort flag, then they contain valid data and processing proceeds to step 16 at which the variable length instruction specified by the bytecodes read is executed. The steps 12, 14 and 16 are performed under hardware control so as to be fast and efficient.

If the determination at step 14 was that any of the bytes being read is marked with a memory abort, then step 18 is initiated which starts a memory abort handler. This memory abort handler can deal with memory aborts of a wide variety of different types, including memory aborts due to fragmentation of the program between discrete memory regions. The rest of this flow diagram illustrates how the memory abort handler deals with such fragmentation induced aborts rather than other types of aborts. It will be appreciated that the abort handler initiated at step 18 could direct processing to other algorithms and mechanisms to deal with different types of memory aborts.

At step 20, the memory abort handler in software controls copying of the current instruction data word containing the bytecode at the start of the current variable length instruction to a fixed location within a fix-up memory region. This is followed at step 22 by copying of a immediately following instruction data word that follows in the sense of the program flow (i.e. is discrete in physical or logical memory space) into an immediately following location within the fix-up memory region. Thus, the two instruction data words are concatenated and placed next to each other within the fix-up memory.

It will be appreciated that the identification of the start of the following memory region could be made in a variety of ways, but can be determined by the memory abort handler such as by examining the memory management unit information or other information specifying, for example, a virtual address to a physical address mapping.

At step 24, the start address of the current variable length instruction which needs to be executed and which spanned the gap between the discrete memory regions is calculated within the fix-up memory region. As an example, if it was known that the current variable length instruction started at the last byte within the instruction data word at the end of the current memory region, then the program counter PC value needed within the fix-up memory region would be pointing to the last byte within the first word of the concatenated data within the fix-up memory.

At step 26, the memory abort handler serves to store in a known location the start address of the following memory region. This start address is needed later in order to determine a correct entry point into the code within the following memory region after the variable length instruction which spans the gap has been executed.

At step 28, a single step flag is set within a register of a Javacard decoder controlling coprocessor CP14. Such control registers are generally conveniently accessible under software control (e.g. coprocessor register store instructions) and can be used by programs to pass configuration information to hardware.

As a last step under the control of the memory abort handler, step 30 serves to generate a state (mode) switching branch instruction BXJ which switches execution from native ARM code in which the memory abort handler is written back to Javacard execution with a program counter value pointing to the start of the current variable length instruction within the fix-up memory. At this point, control is passed back to the hardware.

At step 32, the hardware which has been activated by the BXJ instruction with its associated PC value serves to fill the instruction decoder buffer with two instruction data words taken from the fix-up memory as this is the memory address pointed to by the PC value passed to the hardware. The PC value will also point to the position within the first of the instruction data words where the bytecode of the variable length

Javacard instruction starts and execution of this Javacard variable length instruction is performed at step 34.

The Javacard decoder hardware is responsive to the single step flag set at step 28 to execute only a single Javacard instruction before clearing the single step flag at step 36 and initiating a single step exception handler at step 38.

The single step exception handler passes control again back to software, such as an ARM instruction routine. The single step exception handler serves to restore program flow back to its proper position within the following memory region after the execution of the single instruction from within the fix-up memory region.

At step 40, the single step exception handler serves to read the following memory region address which was stored at step 26 and the program counter PC value which was returned after the hardware had executed the instruction which spanned the gap between discrete memory regions at step 34. It will be appreciated that the hardware updates the PC value itself once it has executed an instruction and accordingly this hardware can effectively be used to indicate the length of the instruction which spanned the gap between the discrete memory regions such that the offset from the start of the following memory region at which the following variable length instruction starts may be determined. More particularly, in some embodiments the least significant bits of the PC value can be used to indicate the offset from the start of the following memory region at which the following variable length instruction will be found.

At step 42, the single step exception handler generates a BXJ instruction switching back to the Javacard execution state with a PC value pointing to the following instruction in the following memory region as was determined at step 40. Processing then returns to step 12.

Figure 4 schematically illustrates the processing described in relation to Figures 3A and 3B.

A current memory region 44 and a discrete following memory region 46 are illustrated. The current memory region 44 terminates with a page table boundary and an attempt to read instruction data from beyond this page table boundary will cause a memory abort. As illustrated, the PC value of the last variable length instruction reached within the current memory region 44 is shown and this current variable length instruction is in fact three bytes in length and accordingly the first two bytes of the following memory region 46 form part of the current variable length instruction.

As illustrated, the end portion of the current memory region 44 constituting the last 32-bit instruction data word of that region, together with the beginning portion of the following memory region 46 constituting the first 32-bit instruction word of that region are both copied under control of the memory abort handler to a fix-up memory region 48 where they are concatenated. The program counter PC from within the current memory region 44 is converted to a program counter PC' within the fix-up memory region 48. A BXJ mode switching branch instruction is then initiated to switch to the Javacard mode and activate the Javacard decoder. The PC' value is passed to the Javacard decoder as an R14 value stored within the normal position for such a BXJ instruction within the general purpose bank of ARM registers as normally addressed in ARM mode.

When the Javacard decoder has been activated it serves to copy the two now concatenated instruction data words from the fix-up memory 48 into the instruction decoder buffer. Execution of the Javacard instruction starting at the program counter value position PC' is then performed and will utilise bytes from both slots within the instruction decoder buffer. Before control was passed to the Javacard decoder by the BXJ instruction, the memory abort handler set a single step flag for the Java decoder which constrains the Java decoder to execute only a single instruction before initiating a single step exception handler software routine. Thus, after the single instruction which spanned the current memory region 44 and following memory region 46 processing passes to the single step exception handler 50. The single step exception handler 50 uses a stored value of the following region start address that was stored in a

predetermined position by the memory abort handler together with the program counter value PCfollowing' that was calculated by the Javacard decoder after execution of its single instruction to together calculate the true PCfollowing value within the following memory region 46. When this has been calculated, another BXJ instruction is generated using the PCfollowing value within the following memory region 46 as a variable to be passed back to the Javacard decoder such that execution of the instructions within the following memory region 46 can commence starting with the first full variable length instruction within that following memory region 46. Thus, the two first instruction data words from the following memory region 46 are copied into the instruction decoder buffer and Javacard execution restarted.

Figure 5 schematically illustrates hardware that may execute the above described techniques. It will be appreciated that Figure 5 is highly simplified and omits for the sake of clarity various circuit elements which are not involved in the present techniques. The system 52 includes a memory 54, a microprocessor execution engine 56, a Javacard decoder 58, an ARM instruction decoder 60, a coprocessor 62 and an instruction decoder buffer 64. The execution of Javacard bytecodes can be performed in the way described in the previously mentioned PCT Published Patent Application WO-A-02/29507. In particular, Javacard bytecodes read from the memory 54 are fed via the instruction decoder buffer 64 to the Javacard decoder 58 where they are used to generate control signals that control the microprocessor execution engine 56 to perform the desired processing. In the ARM mode, the ARM decoder 60 receives ARM instructions from the memory 54 and produces its own control signals to control the microprocessor execution engine 56.

As illustrated, the memory 54 contains various discrete memory regions A and B which store different portions of a Javacard program with variable length instructions being permitted to span the gaps between the memory regions. Also within the memory 54 is a fix-up memory region where concatenation of an end portion and a start portion of discrete memory regions can be made so as to reconstruct the full versions of a variable length instructions which span that gap so that the instruction may be executed out of the fix-up memory region. Storage for information

such as the start address of the following memory region is also provided within the memory 54 for use by the previously described single step exception handler in order to restore processing back to the correct point within the following memory region. Also stored within the memory 54, but not illustrated, will be memory abort handling code and other support code needed by the system 52. This memory abort handling code, support code and the single step exception handler will typically be provided in the form of ARM code to be decoded by the ARM decoder 60.

As illustrated, a coprocessor 62 is coupled to the Javacard decoder 58 and stores various configuration parameters associated with the Javacard decoder 58. Many of these parameters are associated with the initialisation of the Java Virtual Machine upon which the Javacard bytecodes are considered to be executing. As illustrated, one of the registers within the coprocessor 62 contains a flag which serves as a single step flag which when set constrains the Javacard decoder to execute only a single Javacard instruction before triggering the single step exception handler and clearing the single step flag.

The use of this single step flag in connection with the above technique for dealing with memory fragmentation has been discussed, but the single step flag may also be used as a debugging aid even when the Javacard instructions do not span memory regions. Thus, Javacard instructions can be executed one at a time with control then being passed to suitable code serving as the single step exception handler but in fact relating to debug.

It will be appreciated that the above described techniques have been presented in the form of a method for executing an instruction and an apparatus for executing instructions. However, it will be appreciated that a significant portion of the invention is implemented in the form of computer code serving as the memory abort handler and the single step exception hander. This special purpose code may be separately supplied as a commercial item in its own right and can be considered as an embodiment of the invention.

It will be appreciated that the above described embodiments have been directed towards a system which implements the present technique in a mixture of hardware and software. In particular, software is provided in the form of the memory abort handler and the single step exception handler and hardware support for the technique is provided in the form of the single step flag and the responsiveness of the Javacard decoder to the single step flag. It will however be appreciated that other embodiments of the invention may be purely implemented in the form of software.

As an example, detection of a memory abort could be made in the same way and then a memory abort handler triggered which fully dealt with the fragmentation of variable length instructions within software. The first portion of such a software program would provide the functionality of the memory abort handler described above in identifying the memory abort as relating to a variable length instruction being fragmented, and then copying the different portions of that variable length instruction into a fix-up memory region to form a concatenated set of instruction data words containing the entire variable length instruction. In this software implementation, instead of then passing control back to the hardware to execute a single instruction being the variable length instruction, the software could instead emulate this instruction. As part of this emulation a look up table which would specify the length of the variable length instruction depending upon the first byte would be used. This length would be known to the software and accordingly could then be used to calculate the start address within the following memory region of the following variable length instruction. The software could then restore program flow to that following memory region starting at the calculated memory address.

CLAIMS

1. A method of executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of a data processing apparatus, said method comprising the steps of:

detecting an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region;

concatenating instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

diverting program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

restoring program execution flow to execute instructions following said variable length instruction from within said following memory address region.

2. A method as claimed in claim 1, wherein said steps of detecting is performed under hardware control.

3. A method as claimed in any one of claims 1 and 2, wherein said steps of concatenating, diverting and restoring are performed under software control.

4. A method as claimed in any one of claims 1, 2 and 3, wherein variable length instructions are fetched from said memory to an instruction buffer before being executed.

5. A method as claimed in claim 4, wherein said step of detecting occurs as said variable length instruction is read from said instruction buffer.

6. A method as claimed in any one of claims 4 and 5, wherein fetching of variable length instructions to said instruction buffer is performed by fetching instruction data from sequential memory addresses under hardware control.

7. A method as claimed in claim 6, wherein when a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, said buffer memory location is marked as not containing valid instruction data.

8. A method as claimed in claim 7, wherein said step of detecting comprises detecting an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

9. A method as claimed in any one of the preceding claims, wherein a program counter value specifies a location of a variable length instruction to be executed.

10. A method as claimed in claim 9, wherein said steps of diverting and restoring act by modifying said program counter value.

11. A method as claimed in any one of the preceding claims, comprising setting a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

12. A method as claimed in claim 11, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

13. A method as claimed in any one of claims 11 and 12, wherein upon hardware execution of said single variable length instruction, said single step flag is cleared under hardware control.

14. A method as claimed in any one of claims 11, 12 and 13, wherein said single step flag is stored within a coprocessor register.

15. A method as claimed in any one of the preceding claims, comprising calculating a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

16. A method as claimed in claim 15, wherein said step of calculating uses as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

17. A method as claimed in claim 16, comprising storing said start address of said following memory region before said step of diverting.

18. A method as claimed in any one of the preceding claims, wherein said variable length instructions are Javacard bytecode instructions executed as native instructions by said data processing apparatus.

19. A method as claimed in claim 18, wherein said data processing apparatus also supports execution of instructions of a further instruction set, said steps of concatenating, diverting and restoring being performed under control of instructions of said further instruction set.

20. A method as claimed in claim 19, wherein said steps of diverting and restoring are performed using state switching branch instructions that serve to switch to execution of Java bytecodes starting from a specified memory address location.

21. A method as claimed in any one of the preceding claims, wherein a program to be executed is stored within fragmented memory regions within said memory.

22. Apparatus for executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory, said apparatus comprising:

a detector operable to detect an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region;

combining logic operable to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

diverting logic operable to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

restoring logic operable to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

23. Apparatus as claimed in claim 22, wherein said detector is non-programmable hardware.

24. Apparatus as claimed in any one of claims 22 and 23, wherein said concatenating logic, said diverting logic and said restoring logic comprise programmable hardware operating under software control.

25. Apparatus as claimed in any one of claims 22, 23 and 24, wherein variable length instructions are fetched from said memory to an instruction buffer before being executed.

26. Apparatus as claimed in claim 25, wherein said detector acts as said variable length instruction is read from said instruction buffer.

27. Apparatus as claimed in any one of claims 25 and 26, wherein fetching of variable length instructions to said instruction buffer is performed by fetching instruction data from sequential memory addresses under hardware control.

28. Apparatus as claimed in claim 27, wherein when a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, said buffer memory location is marked as not containing valid instruction data.

29. Apparatus as claimed in claim 28, wherein said detector is operable to detect an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

30. Apparatus as claimed in any one of claims 22 to 29, wherein a program counter value specifies a location of a variable length instruction to be executed.

31. Apparatus as claimed in claim 30, wherein said diverting logic and said restoring logic act by modifying said program counter value.

32. Apparatus as claimed in any one claims 22 to 31, comprising setting a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

33. Apparatus as claimed in claim 32, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

34. Apparatus as claimed in any one of claims 32 and 33, wherein upon hardware execution of said single variable length instruction, said single step flag is cleared under hardware control.

35. Apparatus as claimed in any one of claims 32, 33 and 34, wherein said single step flag is stored within a coprocessor register.

36. Apparatus as claimed in any one claims 22 to 35, comprising calculating logic operable to calculate a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

37. Apparatus as claimed in claim 36, wherein said calculating logic uses as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

38. Apparatus as claimed in claim 37, comprising storing said start address of said following memory region before said diversion.

39. Apparatus as claimed in any one of claims 22 to 38, wherein said variable length instructions are Javacard bytecode instructions executed as native instructions by said data processing apparatus.

40. Apparatus as claimed in claim 39, wherein said data processing apparatus also supports execution of instructions of a further instruction set, said steps of concatenating, diverting and restoring being performed under control of instructions of said further instruction set.

41. Apparatus as claimed in claim 40, wherein said diverting logic and said restoring logic use mode switching branch instructions that serve to switch to execution of Java bytecodes starting from a specified memory address location.

42. Apparatus as claimed in any one of claims 22 to 41, wherein a program to be executed is stored within fragmented memory regions within said memory.

43. A computer program product for controlling a data processing apparatus operable to executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of said data processing apparatus, said computer program product comprising:

code operable after an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region, said code including:

(i) concatenating code operable to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

(ii) diverting code operable to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

(iii) restoring code operable to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

44. A computer program product as claimed in claim 43, wherein detection of said attempt to execute a variable length instruction spanning two discrete memory address regions is performed under hardware control.

45. A computer program product as claimed in any one of claims 43 and 44, wherein variable length instructions are fetched from said memory to an instruction buffer before being executed.

46. A computer program product as claimed in claim 45, wherein said detection occurs as said variable length instruction is read from said instruction buffer.

47. A computer program product as claimed in any one of claims 45 and 46, wherein fetching of variable length instructions to said instruction buffer is performed

by fetching instruction data from sequential memory addresses under hardware control.

48. A computer program product as claimed in claim 47, wherein when a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, said buffer memory location is marked as not containing valid instruction data.

49. A computer program product as claimed in claim 48, wherein said detection comprises detecting an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

50. A computer program product as claimed in any one of claims 43 to 49, wherein a program counter value specifies a location of a variable length instruction to be executed.

51. A computer program product as claimed in claim 50, wherein said diverting code and said restoring code act by modifying said program counter value.

52. A computer program product as claimed in any one of claims 43 to 51, comprising setting code operable to set a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

53. A computer program product as claimed in claim 52, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

54. A computer program product as claimed in any one of claims 52 and 53, wherein upon hardware execution of said single variable length instruction, said single step flag is cleared under hardware control.

55. A computer program product as claimed in any one of claims 52, 53 and 54, wherein said single step flag is stored within a coprocessor register.

56. A computer program product as claimed in any one of claims 43 to 55, comprising calculating code operable to calculate a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

57. A computer program product as claimed in claim 56, wherein said calculating code uses as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

58. A computer program product as claimed in claim 57, comprising storing said start address of said following memory region before said diversion.

59. A computer program product as claimed in any one of claims 43 to 58, wherein said variable length instructions are Javacard bytecode instructions executed as native instructions by said data processing apparatus.

60. A computer program product as claimed in claim 59, wherein said data processing apparatus also supports execution of instructions of a further instruction set, said steps of concatenating, diverting and restoring being performed under control of instructions of said further instruction set.

61. A computer program product as claimed in claim 60, wherein said diverting code and said restoring code use state switching branch instructions that serve to

switch to execution of Java bytecodes starting from a specified memory address location.

62. A computer program product as claimed in any one of claims 43 to 61, wherein a program to be executed is stored within fragmented memory regions within said memory.

63. A method substantially as hereinbefore described with reference to the accompanying drawings.

64. Apparatus substantially as hereinbefore described with reference to the accompanying drawings.

65. A computer program product substantially as hereinbefore described with reference to the accompanying drawings.

ABSTRACT

EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

Within a system supporting execution of variable length instructions a program is stored within discrete memory regions with a variable length instruction spanning a gap between two such discrete memory regions. When execution is attempted of such a variable length instruction spanning a gap, an abort handler is initiated which serves to copy the end portion of one of the memory regions together with the start portion of the other memory region into a separate fix-up memory region where these may be concatenated such that the whole of the variable length instruction will appear in one place. Execution of that variable length instruction from out of the fix-up memory region can then be triggered. This execution is constrained by the setting of a single step flag which causes the hardware to only execute the single instruction which span the gap before returning control to a single step exception handler which can then restore program flow to the point in the following memory region after the variable length instruction which spanned the gap.

[Figure 4]

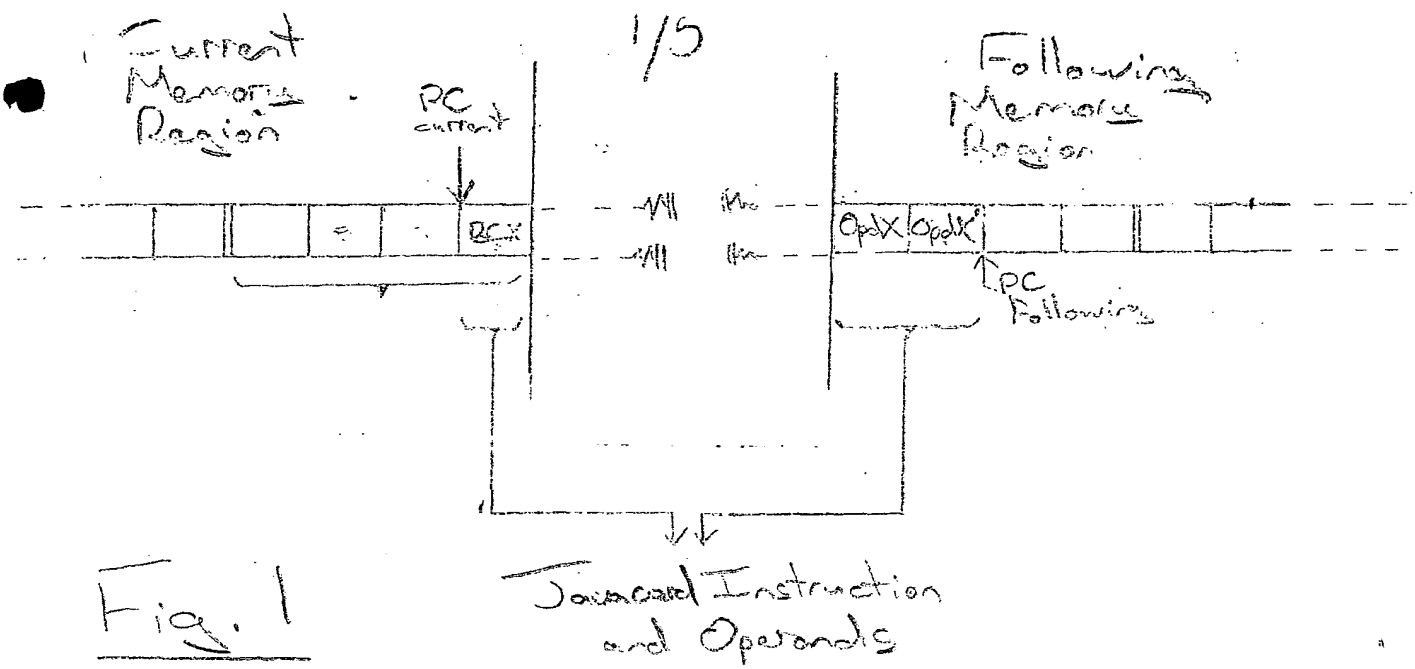

Current Memory Region
PC current
Following Memory Region
PC Following
Javacard Instruction and Operands

Fig. 1

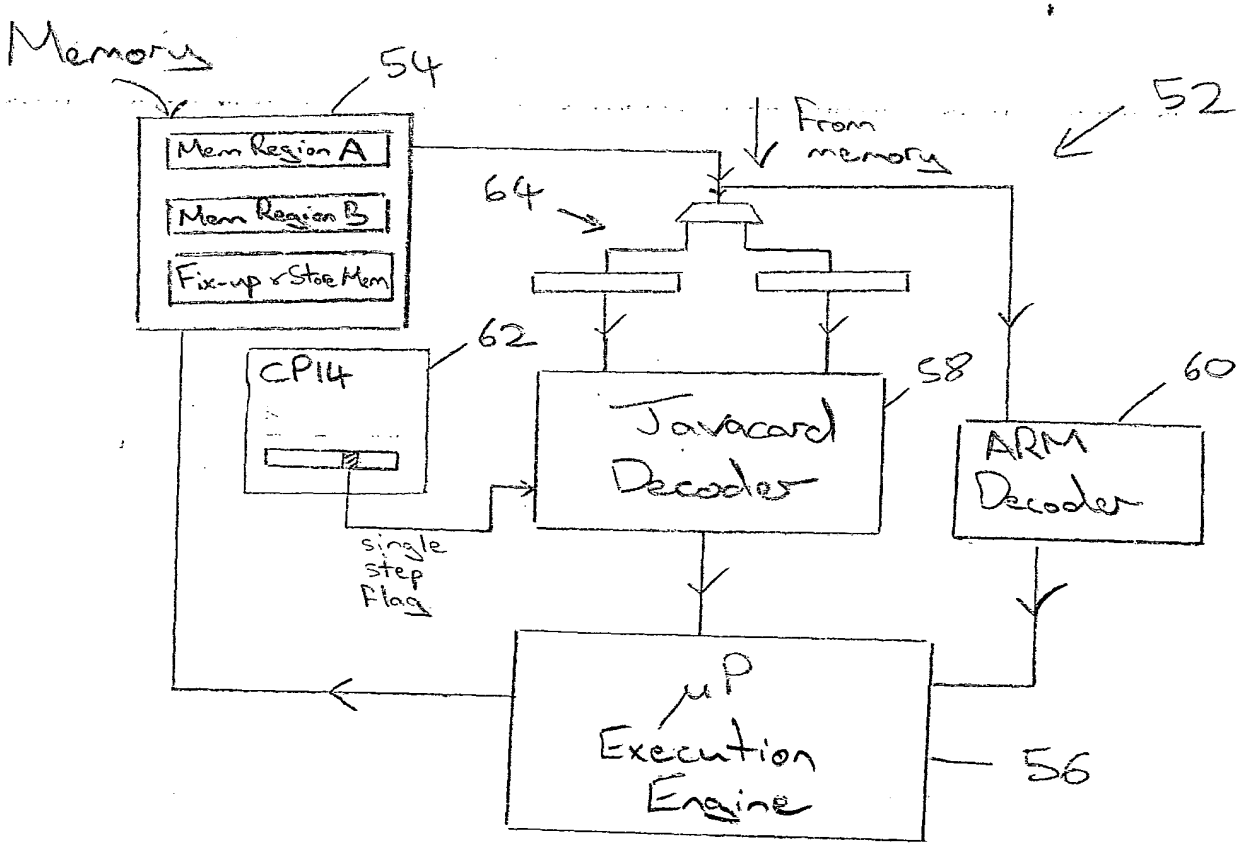

Memory
54
Mem Region A
Mem Region B
Fix-up + Stor Mem
From memory
52
64
62
CP14
58
Javacard Decoder
60
ARM Decoder
single step Flag
µP Execution Engine
56

Fig. 5

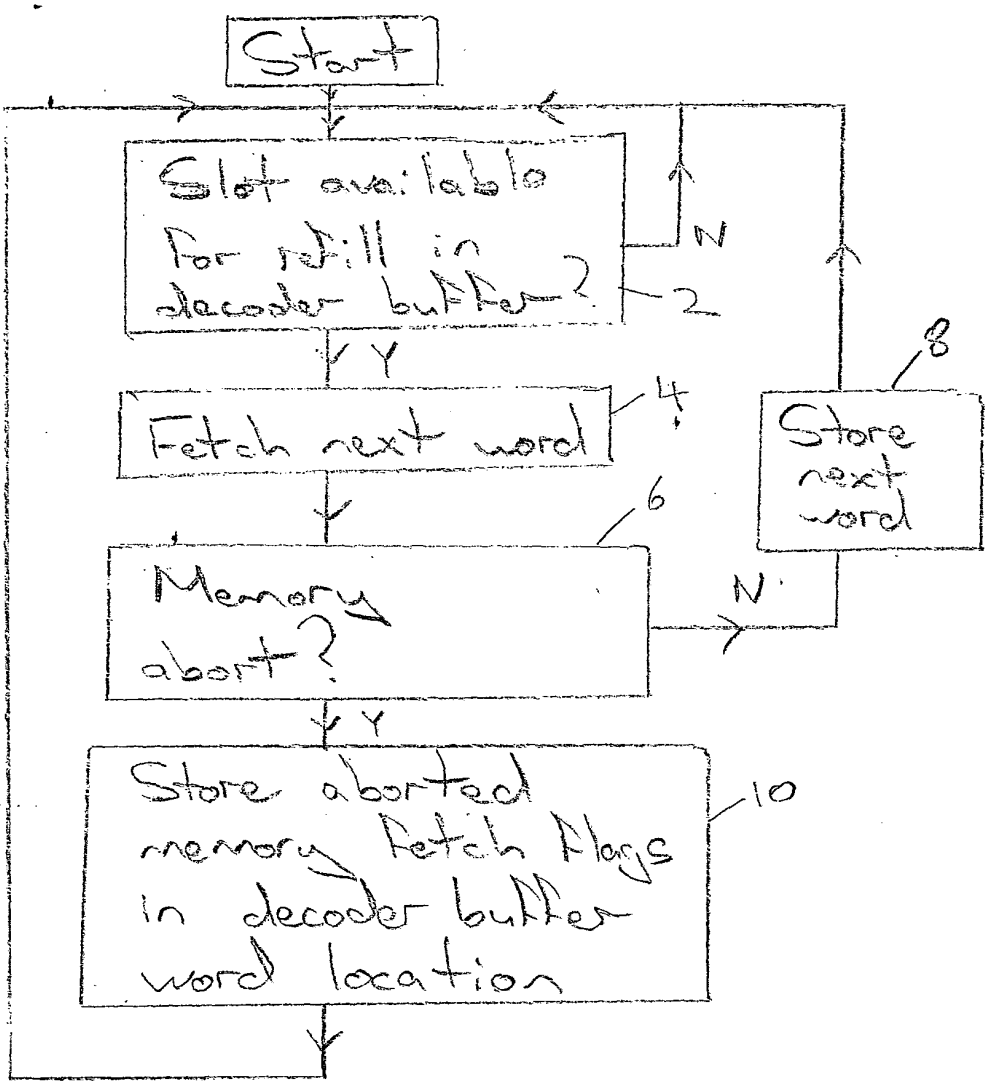

Start
Slot available for refill in decoder buffer?
2
N
Y
Fetch next word
4
Store next word
8
Memory abort?
6
N
Y
Store aborted memory fetch flags in decoder buffer word location
10

Fig. 2

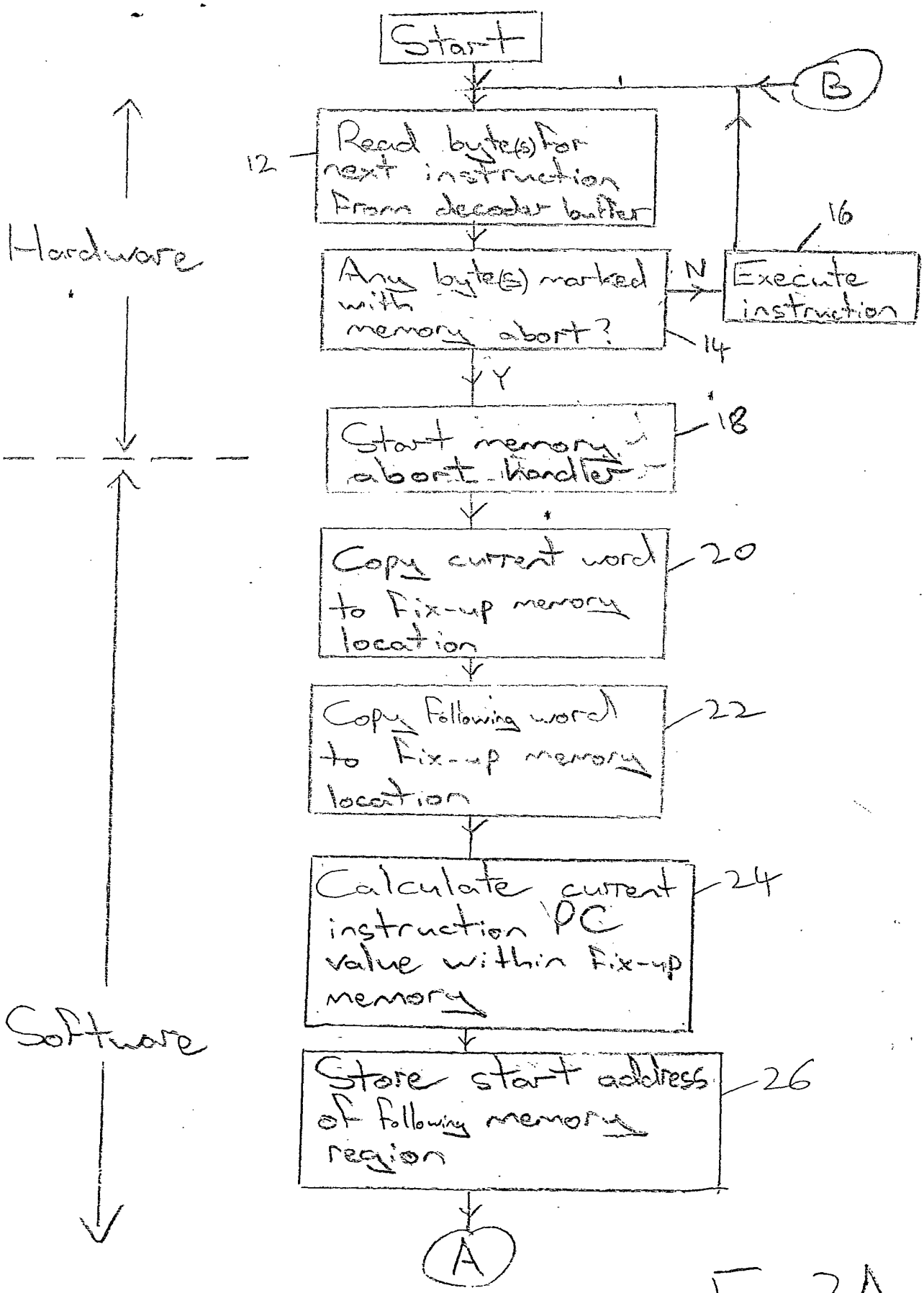

Start
B
Read byte(s) for next instruction From decoder buffer
12
Hardware
Any byte(s) marked with memory abort?
N
Execute instruction
16
14
Y
Start memory abort handler
18
Copy current word to Fix-up memory location
20
Copy following word to Fix-up memory location
22
Calculate current instruction PC value within Fix-up memory
24
Software
Store start address of following memory region
26
A

Fig. 3A

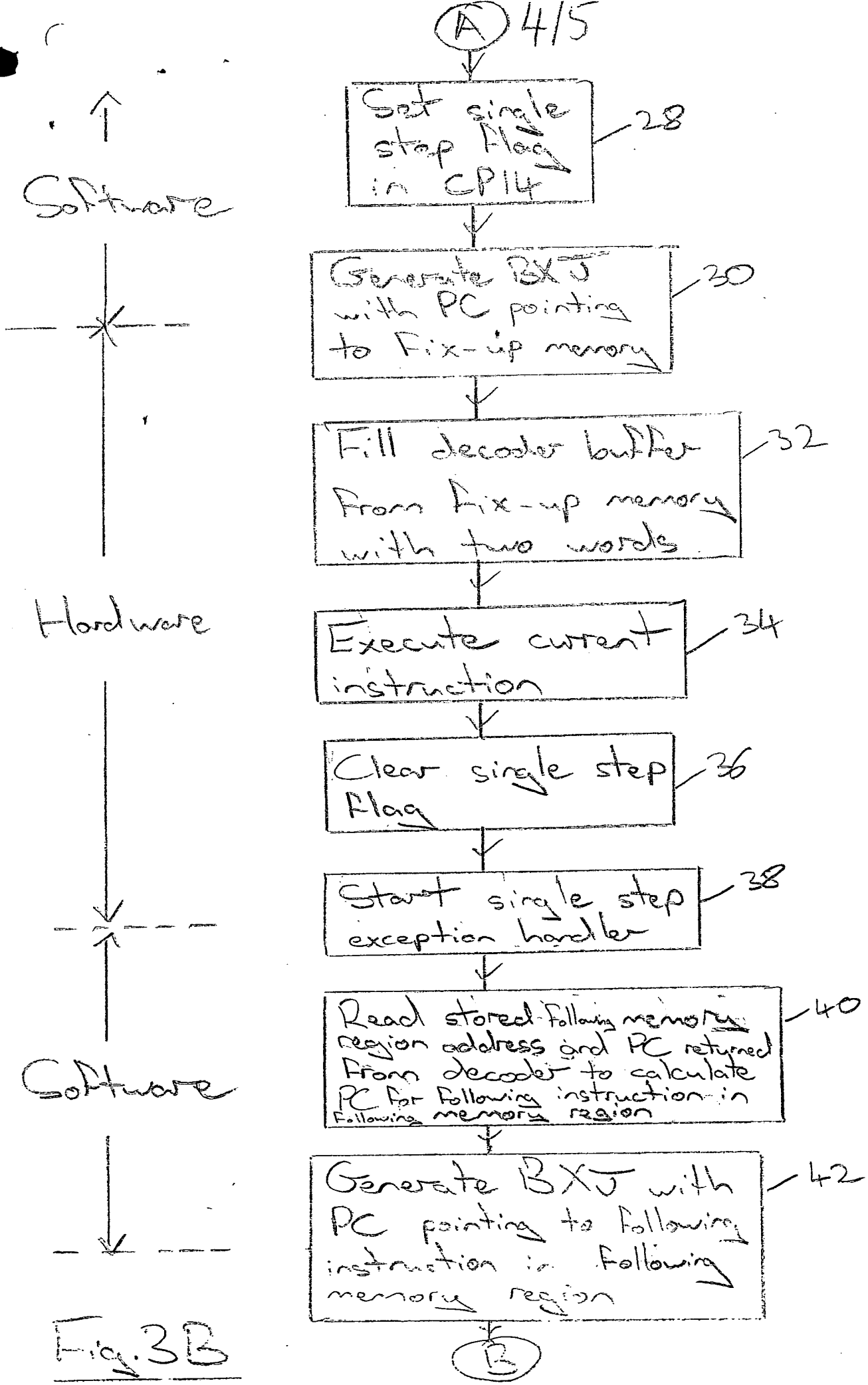

A
Set single step Flag in CPI4
28
Generate BXJ with PC pointing to Fix-up memory
30
Fill decoder buffer From Fix-up memory with two words
32
Execute current instruction
34
Clear single step Flag
36
Start single step exception handler
38
Read stored following memory region address and PC returned From decoder to calculate PC for following instruction in Following memory region
40
Generate BXJ with PC pointing to following instruction in following memory region
42
B
Software
Hardware
Software

Fig. 3B

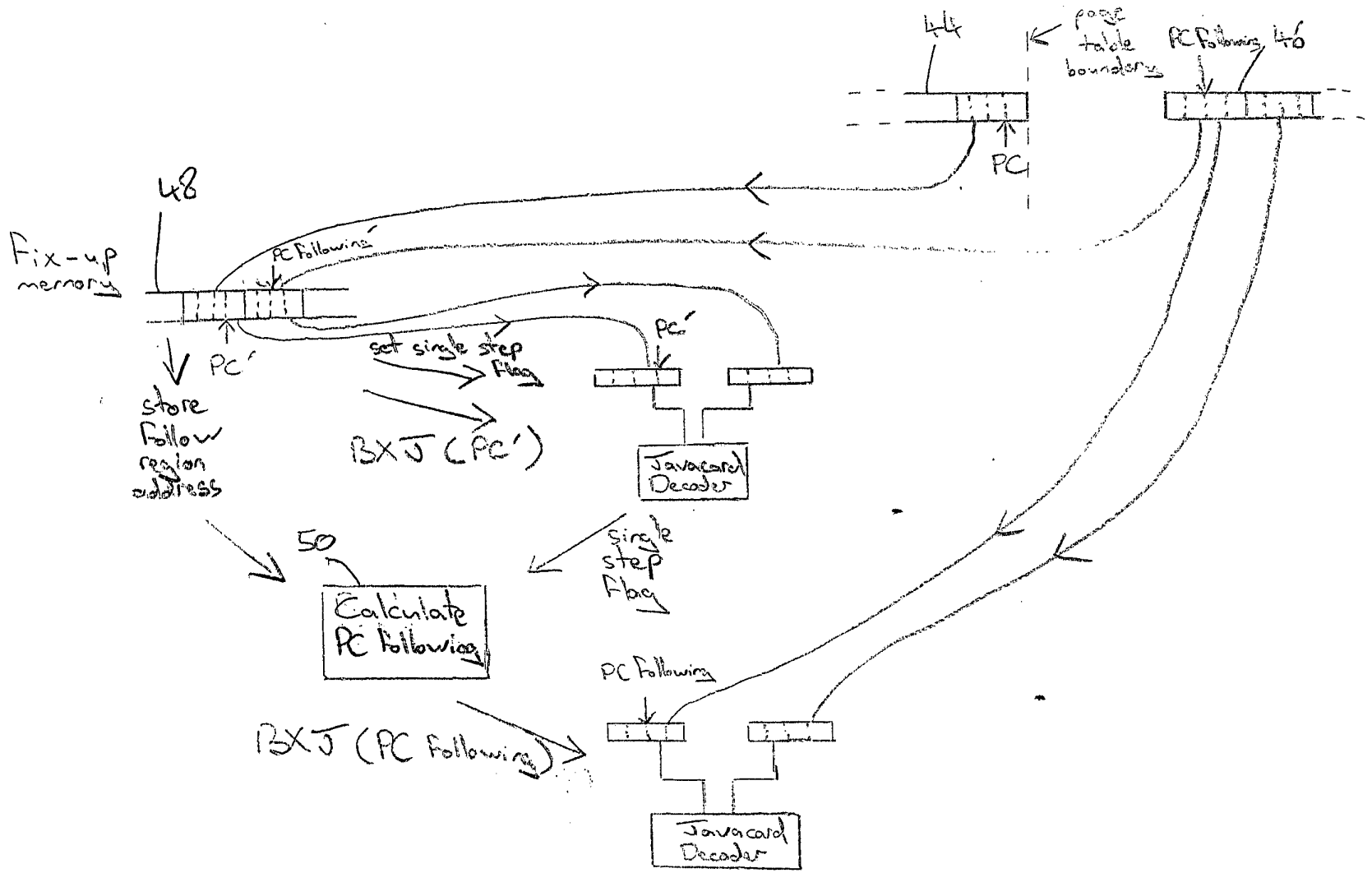

44
page table boundary
PC Following 46
PC
48
Fix-up memory
PC Following
PC'
set single step flag
PC'
store follow region address
BXJ (PC')
Javacard Decoder
50
single step flag
Calculate PC following
PC Following
BXJ (PC Following)
Javacard Decoder

Fig. 4